HTTPbis Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 05, 2014

R. Peon
Google, Inc
H. Ruellan
Canon CRF

April 03, 2014

HPACK - Header Compression for HTTP/2
draft-ietf-httpbis-header-compression-07

Abstract

This specification defines HPACK, a compression format for efficiently representing HTTP header fields in the context of HTTP/2.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

Working Group information can be found at <http://tools.ietf.org/wg/httpbis/>; that specific to HTTP/2 are at <http://http2.github.io/>.

The changes in this draft are summarized in Appendix A.1.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 05, 2014.

Table of Contents

1. Introduction

This specification defines HPACK, a compression format for efficiently representing HTTP header fields in the context of HTTP/2 (see [HTTP2]).

2. Overview

In HTTP/1.1 (see [HTTP-p1]), header fields are encoded without any form of compression. As web pages have grown to include dozens to hundreds of requests, the redundant header fields in these requests now measurably increase latency and unnecessarily consume bandwidth (see [PERF1] and [PERF2]).

SPDY [SPDY] initially addressed this redundancy by compressing header fields using the DEFLATE format [DEFLATE], which proved very effective at efficiently representing the redundant header fields. However, that approach exposed a security risk as demonstrated by the CRIME attack (see [CRIME]).

This document describes HPACK, a new compressor for header fields which eliminates redundant header fields, is not vulnerable to known security attacks, and which also has a bounded memory requirement for use in constrained environments.

2.1. Outline

The HTTP header field encoding defined in this document is based on a header table that maps name-value pairs to index values. The header table is incrementally updated during the HTTP/2 connection.

A set of header fields is treated as an unordered collection of name-value pairs. Names and values are considered to be opaque sequences of octets. The order of header fields is not guaranteed to be preserved after being compressed and decompressed.

As two consecutive sets of header fields often have header fields in common, each set is coded as a difference from the previous set. The goal is to only encode the changes (header fields present in one of the sets that are absent from the other) between the two sets of header fields.

A header field is represented either literally or as a reference to a name-value pair in the header table. A set of header fields is stored as a set of references to entries in the header table (possibly keeping only a subset of it, as some header fields may be missing a corresponding entry in the header table). Differences between consecutive sets of header fields are encoded as changes to the set of references.

The encoder is responsible for deciding which header fields to insert as new entries in the header table. The decoder executes the modifications to the header table and reference set prescribed by the encoder, reconstructing the set of header fields in the process. This enables decoders to remain simple and understand a wide variety of encoders.

Examples illustrating the use of these different mechanisms to represent header fields are available in Appendix D.

3. Header Field Encoding

3.1. Encoding Concepts

The encoding and decoding of header fields relies on some components and concepts:

Header Field: A name-value pair. Both the name and value are treated as opaque sequences of octets.

Header Table: The header table (see Section 3.1.2) is a component used to associate stored header fields to index values.

Static Table: The static table (see Appendix B) is a component used to associate static header fields to index values. This data is ordered, read-only, always accessible, and may be shared amongst all encoding contexts.

Reference Set: The reference set (see Section 3.1.3) is a component containing an unordered set of references to entries in the header table. This is used for the differential encoding of a new header set.

Header Set: A header set is an unordered group of header fields that are encoded jointly. A complete set of key-value pairs contained in a HTTP request or response is a header set.

Header Field Representation: A header field can be represented in encoded form either as a literal or as an index (see Section 3.1.4).

Header Block: The entire set of encoded header field representations which, when decoded, yield a complete header set.

Header Field Emission: When decoding a set of header field representations, some operations emit a header field (see Section 3.1.5). Emitted header fields are added to the current header set and cannot be removed.

3.1.1. Encoding Context

The set of mutable structures used within an encoding context include a header table and a reference set. Everything else is either immutable or conceptual.

HTTP messages are exchanged between a client and a server in both directions. The encoding of header fields in each direction is independent from the other direction. There is a single encoding context for each direction used to encode all header fields sent in that direction.

3.1.2. Header Table

A header table consists of a list of header fields maintained in first-in, first-out order. The first and newest entry in a header table is always at index 1, and the oldest entry of a header table is at the index len(header table).

The header table is initially empty.

There is typically no need for the header table to contain duplicate entries. However, duplicate entries MUST NOT be treated as an error by a decoder.

The encoder decides how to update the header table and as such can control how much memory is used by the header table. To limit the memory requirements of the decoder, the header table size is strictly bounded (see Section 3.3.1).

The header table is updated during the processing of a set of header field representations (see Section 3.2.1).

3.1.3. Reference Set

A reference set is an unordered set of references to entries of the header table.

The reference set is initially empty.

The reference set is updated during the processing of a set of header field representations (see Section 3.2.1).

The reference set enables differential encoding, whereby only differences between the previous header set and the current header set need to be encoded. The use of differential encoding is optional for any header set.

When an entry is evicted from the header table, if it was referenced from the reference set, its reference is removed from the reference set.

To limit the memory requirements on the decoder side for handling the reference set, only entries within the header table can be contained in the reference set. To still allow entries from the static table to take advantage of the differential encoding, when a header field is represented as a reference to an entry of the static table, this entry is inserted into the header table (see Section 3.2.1).

3.1.4. Header Field Representation

An encoded header field can be represented either as a literal or as an index.

Literal Representation: A literal representation defines a new header field. The header field name is represented either literally or as a reference to an entry of the header table. The header field value is represented literally.

Three different literal representations are provided:

* A literal representation that does not add the header field to the header table (see Section 4.3.2).

* A literal representation that does not add the header field to the header table and require that this header field always use a literal representation, in particular when re-encoded by an intermediary (see Section 4.3.3).

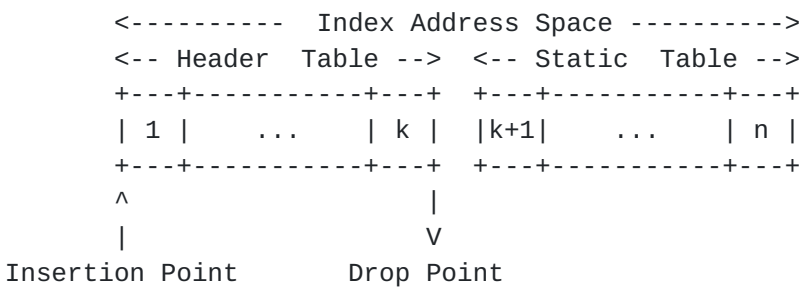
* A literal representation that adds the header field as a new entry at the beginning of the header table (see Section 4.3.1).

Indexed Representation: The indexed representation defines a header field as a reference to an entry in either the header table or the static table (see Section 4.2).

Indices between 1 and len(header table), inclusive, refer to elements in the header table, with index 1 referring to the beginning of the table.

Indices between len(header table) + 1 and len(header table) + len(static table), inclusive, refer to elements in the static table, where the index len(header table) + 1 refers to the first entry in the static table.

Any other indices MUST be treated as a decoding error.



Index Address Space

3.1.5. Header Field Emission

The emission of a header field is the process of marking a header field as belonging to the current header set. Once a header has been emitted, it cannot be removed from the current header set.

On the decoding side, an emitted header field can be safely passed to the upper processing layer as part of the current header set. The decoder MAY pass the emitted header fields to the upper processing layer in any order.

By emitting header fields instead of emitting header sets, the decoder can be implemented in a streaming way, and as such has only to keep in memory the header table and the reference set. This bounds the amount of memory used by the decoder, even in presence of a very large set of header fields. The management of memory for handling very large sets of header fields can therefore be deferred to the upper processing layers.

3.2. Header Block Decoding

The processing of a header block to obtain a header set is defined in this section. To ensure that the decoding will successfully produce a header set, a decoder MUST obey the following rules.

3.2.1. Header Field Representation Processing

All the header field representations contained in a header block are processed in the order in which they are presented, as specified below.

An _indexed representation_ with an index value of 0 entails one of the following actions, depending on what is encoded next:

- The reference set is emptied.

- The maximum size of the header table is updated.

An _indexed representation_ corresponding to an entry _present_ in the reference set entails the following actions:

- The entry is removed from the reference set.

An _indexed representation_ corresponding to an entry _not present_ in the reference set entails the following actions:

- If referencing an element of the static table:
  - The header field corresponding to the referenced entry is emitted.
  - The referenced static entry is inserted at the beginning of the header table.
  - A reference to this new header table entry is added to the reference set, except if this new entry didn't fit in the header table.
- If referencing an element of the header table:
  - The header field corresponding to the referenced entry is emitted.
  - The referenced header table entry is added to the reference set.

A _literal representation_ that is _not added_ to the header table entails the following action:

- The header field is emitted.

A _literal representation_ that is _added_ to the header table entails the following actions:

- The header field is emitted.
- The header field is inserted at the beginning of the header table.
- A reference to the new entry is added to the reference set (except if this new entry didn't fit in the header table).

3.2.2. Reference Set Emission

Once all the representations contained in a header block have been processed, the header fields referenced in the reference set which have not previously been emitted during this processing are emitted.

3.2.3. Header Set Completion

Once all of the header field representations have been processed, and the remaining items in the reference set have been emitted, the header set is complete.

3.3. Header Table Management

3.3.1. Maximum Table Size

To limit the memory requirements on the decoder side, the size of the header table is bounded. The size of the header table MUST stay lower than or equal to its maximum size.

By default, the maximum size of the header table is equal to the value of the HTTP/2 setting SETTINGS_HEADER_TABLE_SIZE defined by the decoder (see [HTTP2]). The encoder can change this maximum size (see Section 4.4), but it must stay lower than or equal to the value of SETTINGS_HEADER_TABLE_SIZE.

The size of the header table is the sum of the size of its entries.

The size of an entry is the sum of its name's length in octets (as defined in Section 4.1.2), of its value's length in octets (Section 4.1.2) and of 32 octets.

The lengths are measured on the non-encoded entry name and entry value (for the case when a Huffman encoding is used to transmit string values).

The 32 octets are an accounting for the entry structure overhead. For example, an entry structure using two 64-bits pointers to reference the name and the value and the entry, and two 64-bits integer for counting the number of references to these name and value would use 32 octets.

3.3.2. Entry Eviction When Header Table Size Changes

Whenever an entry is evicted from the header table, any reference to that entry contained by the reference set is removed.

Whenever the maximum size for the header table is made smaller, entries are evicted from the end of the header table until the size of the header table is less than or equal to the maximum size.

The eviction of an entry from the header table causes the index of the entries in the static table to be reduced by one.

3.3.3. Entry Eviction when Adding New Entries

Whenever a new entry is to be added to the table, any name referenced by the representation of this new entry is cached, and then entries are evicted from the end of the header table until the size of the header table is less than or equal to (maximum size - new entry size), or until the table is empty.

If the size of the new entry is less than or equal to the maximum size, that entry is added to the table. It is not an error to attempt to add an entry that is larger than the maximum size.

4. Detailed Format

4.1. Low-level representations

4.1.1. Integer representation

Integers are used to represent name indexes, pair indexes or string lengths. To allow for optimized processing, an integer representation always finishes at the end of an octet.

An integer is represented in two parts: a prefix that fills the current octet and an optional list of octets that are used if the integer value does not fit within the prefix. The number of bits of the prefix (called N) is a parameter of the integer representation.

The N-bit prefix allows filling the current octet. If the value is small enough (strictly less than 2^N-1), it is encoded within the N-bit prefix. Otherwise all the bits of the prefix are set to 1 and the value is encoded using an unsigned variable length integer representation (see <http://en.wikipedia.org/wiki/Variable-length_quantity>). N is always between 1 and 8 bits. An integer starting at an octet-boundary will have an 8-bit prefix.

The algorithm to represent an integer I is as follows:

```
if I < 2^N - 1, encode I on N bits
else
    encode (2^N - 1) on N bits
    I = I - (2^N - 1)
```

```
    while I >= 128
         encode (I % 128 + 128) on 8 bits
         I = I / 128
    encode I on 8 bits
```

For informational purpose, the algorithm to decode an integer I is as follows:

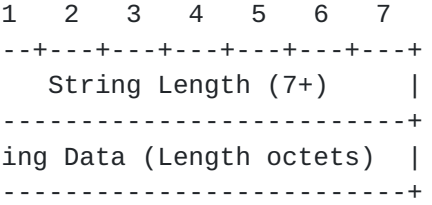
```
decode I from the next N bits
if I < 2^N - 1, return I
else
    M = 0
    repeat
        B = next octet
        I = I + (B & 127) * 2^M
        M = M + 7
    while B & 128 == 128
    return I
```

Examples illustrating the encoding of integers are available in Appendix D.1.

This integer representation allows for values of indefinite size. It is also possible for an encoder to send a large number of zero values, which can waste octets and could be used to overflow integer values. Excessively large integer encodings - in value or octet length - MUST be treated as a decoding error. Different limits can be set for each of the different uses of integers, based on implementation constraints.

4.1.2. String Literal Representation

Header field names and header field values can be represented as literal string. A literal string is encoded as a sequence of octets, either by directly encoding the literal string's octets, or by using a canonical [CANON] Huffman encoding [HUFF].



String Literal Representation

A literal string representation contains the following fields:

H: A one bit flag, H, indicating whether or not the octets of the string are Huffman encoded.

String Length: The number of octets used to encode the string literal, encoded as an integer with 7-bit prefix (see Section 4.1.1).

String Data: The encoded data of the string literal. If H is '0', then the encoded data is the raw octets of the string literal. If H is '1', then the encoded data is the Huffman encoding of the string literal.

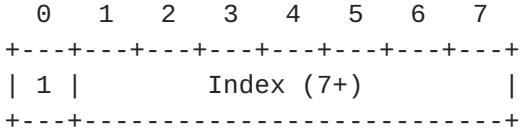
String literals which use Huffman encoding are encoded with the Huffman codes defined in Appendix C (see examples inRequest Examples with Huffman Appendix D.4 and in Response Examples with Huffman Appendix D.6). The encoded data is the bitwise concatenation of the Huffman codes corresponding to each octet of the string literal.

As the Huffman encoded data doesn't always end at an octet boundary, some padding is inserted after it up to the next octet boundary. To prevent this padding to be misinterpreted as part of the string literal, the most significant bits of the EOS (end-of-string) entry in the Huffman table are used.

Upon decoding, an incomplete Huffman code at the end of the encoded data is to be considered as padding and discarded. A padding strictly longer than 7 bits MUST be treated as a decoding error. A padding not corresponding to the most significant bits of the EOS entry MUST be treated as a decoding error. A Huffman encoded string literal containing the EOS entry MUST be treated as a decoding error.

4.2. Indexed Header Field Representation

An indexed header field representation either identifies an entry in the header table or static table. The processing of an indexed header field representation is described in Section 3.2.1.



Indexed Header Field

This representation starts with the '1' 1-bit pattern, followed by the index of the matching pair, represented as an integer with a 7-bit prefix.

The index value of 0 is not used. It MUST be treated as a decoding error if found in an indexed header field representation.

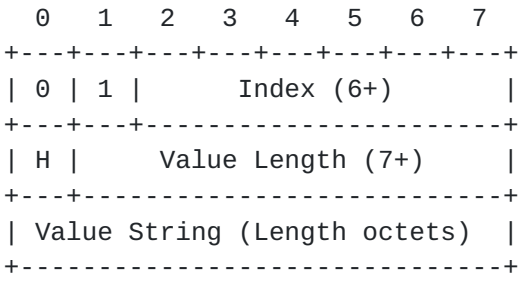
4.3. Literal Header Field Representation

Literal header field representations contain a literal header field value. Header field names are either provided as a literal or by reference to an existing header table or static table entry.

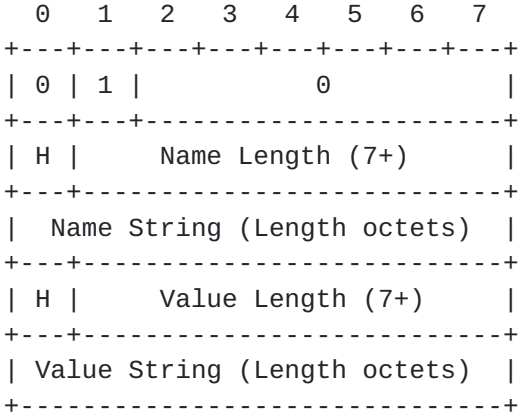
Literal representations all result in the emission of a header field when decoded.

4.3.1. Literal Header Field with Incremental Indexing

A literal header field with incremental indexing adds a new entry to the header table.



Literal Header Field with Incremental Indexing - Indexed Name



Literal Header Field with Incremental Indexing - New Name

This representation starts with the '01' 2-bit pattern.

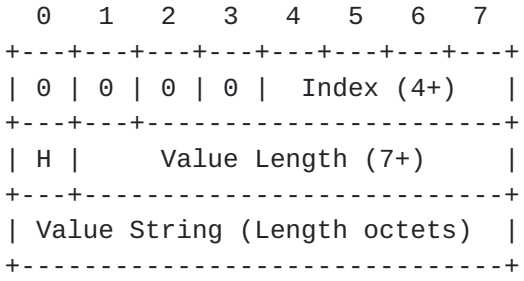
If the header field name matches the header field name of a (name, value) pair stored in the Header Table or Static Table, the header field name can be represented using the index of that entry. In this case, the index of the entry, index (which is strictly greater than 0), is represented as an integer with a 6-bit prefix (see Section 4.1.1).

Otherwise, the header field name is represented as a literal. The value 0 is represented on 6 bits followed by the header field name (see Section 4.1.2).

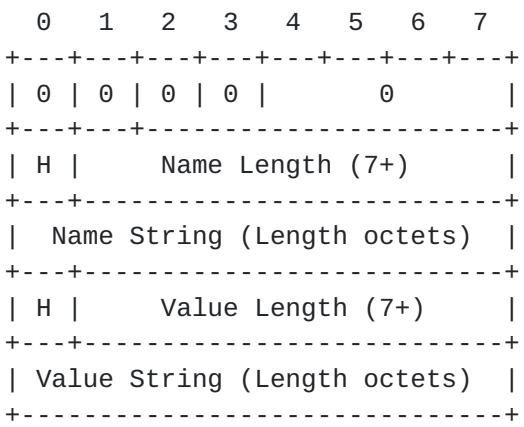
The header field name representation is followed by the header field value represented as a literal string as described in Section 4.1.2.

4.3.2. Literal Header Field without Indexing

A literal header field without indexing causes the emission of a header field without altering the header table.



Literal Header Field without Indexing - Indexed Name



Literal Header Field without Indexing - New Name

The literal header field without indexing representation starts with the '0000' 4-bit pattern.

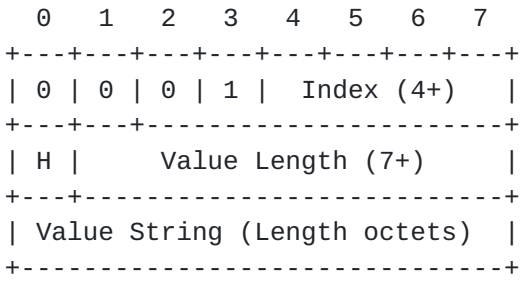
If the header field name matches the header field name of a (name, value) pair stored in the Header Table or Static Table, the header field name can be represented using the index of that entry. In this case, the index of the entry, index (which is strictly greater than 0), is represented as an integer with a 6-bit prefix (see Section 4.1.1).

Otherwise, the header field name is represented as a literal. The value 0 is represented on 4 bits followed by the header field name (see Section 4.1.2).

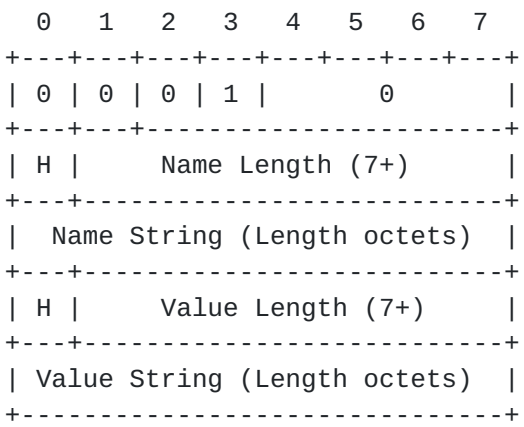
The header field name representation is followed by the header field value represented as a literal string as described in Section 4.1.2.

4.3.3. Literal Header Field never Indexed

A literal header field never indexed causes the emission of a header field without altering the header table.



Literal Header Field never Indexed - Indexed Name



Literal Header Field never Indexed - New Name

The literal header field never indexed representation starts with the '0001' 4-bit pattern.

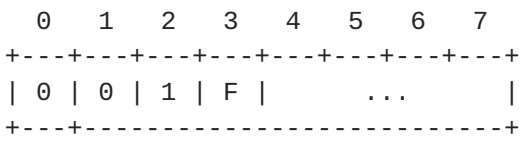
When a header field is represented as a literal header field never indexed, it MUST always be encoded with this same representation. In particular, when a peer sends a header field that it received represented as a literal header field never indexed, it MUST use the same representation to forward this header field.

This representation is intended for protecting header field values that are not to be put at risk by compressing them (see Section 5.1 for more details).

The encoding of the representation is the same as for the literal header field without indexing representation (see Section 4.3.2).

4.4. Encoding Context Update

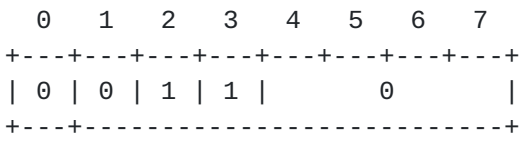
An encoding context update causes the immediate application of a change to the encoding context.



Context Update

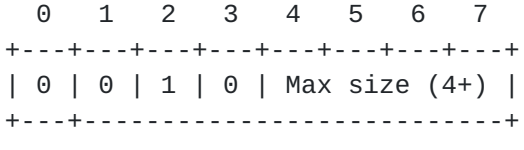
An encoding context update starts with the '001' 3-bit pattern.

It is followed by a flag specifying the type of the change, and by any data necessary to describe the change itself.



Reference Set Emptying

The flag bit being set to '1' signals that the reference set is emptied. The remaining bits are set to '0'.



Maximum Header Table Size Change

The flag bit being set to '0' signals that a change to the maximum size of the header table. This new maximum size MUST be lower than or equal to the value of the setting SETTINGS_HEADER_TABLE_SIZE (see [HTTP2]).

The new maximum size is encoded as an integer with a 4-bit prefix.

Change in the maximum size of the header table can trigger entry evictions (see Section 3.3.2).

5. Security Considerations

5.1. Compression-based Attacks

Compression can create a weak point allowing an attacker to recover secret data. For example, the CRIME attack (see [CRIME]) took advantage of the DEFLATE mechanism (see [DEFLATE]) of SPDY (see [SPDY]) to efficiently probe the compression context. The full-text compression mechanism of DEFLATE allowed the attacker to learn some information from each failed attempt at guessing the secret.

For this reason, HPACK provides only limited compression mechanisms in the form of an indexing table and of a static Huffman encoding.

The indexing table can still provide information to an attacker that would be able to probe the compression context. However, this information is limited to the knowledge of whether the attacker's guess is correct or not.

Still, an attacker could take advantage of this limited information for breaking low-entropy secrets using a brute-force attack. A server usually has some protections against such brute-force attack. Here, the attack would target the client, where it would be harder to detect. The attack would be even more dangerous if the attacker is able to prevent the traffic generated by its brute-force attack from reaching the server.

To offer some protection against such type of attacks, HPACK enables an endpoint to indicate that a header field must never be compressed, across any hop up to the other endpoint (see Section 4.3.3). An endpoint MUST use this feature to prevent the compression of any header field whose value contains a secret which could be put at risk by a brute-force attack.

For optimal processing, a sensitive value (for example a cookie) needs to have an entropy high enough to not be endangered by a brute-force attack, in order to take advantage of HPACK indexing.

There is currently no known threat taking advantage of the use of a fixed Huffman encoding. A study has shown that using a fixed Huffman encoding table created an information leakage, however this same study concluded that an attacker could not take advantage of this information leakage to recover any meaningful amount of information (see [PETAL]).

5.2. Memory Consumption

An attacker can try to cause an endpoint to exhaust its memory. HPACK is designed to limit both the peak and state amounts of memory allocated by an endpoint.

The amount of memory used by the compressor state is limited by the value of the setting SETTINGS_HEADER_TABLE_SIZE. This limitation takes into account both the size of the data stored in the header table, and the overhead required by the table structure itself.

For the decoding side, an endpoint can limit the amount of state memory used by setting an appropriate value for SETTINGS_HEADER_TABLE_SIZE. For the encoding side, the endpoint can limit the amount of state memory it uses by defining a header table maximum size lower than the value of SETTINGS_HEADER_TABLE_SIZE defined by its peer (see Section 4.4).

The amount of temporary memory consumed is linked to the set of header fields emitted or received. However, this amount of temporary memory can be limited by processing these header fields in a streaming manner.

5.3. Implementation Limits

An implementation of HPACK needs to ensure that large values for integers, long encoding for integers, or long string literal do not create security weaknesses.

An implementation has to set a limit for the values it accepts for integers, as well as for the encoded length (see Section 4.1.1). In the same way, it has to set a limit to the length it accepts for string literals (see Section 4.1.2).

6. Acknowledgements

This document includes substantial editorial contributions from the following individuals: Mike Bishop, Jeff Pinner, Julian Reschke, Martin Thomson.

7. References

7.1. Normative References

[HTTP-p1] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-26 (work in progress), February 2014.

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-10 (work in progress), February 2014.

7.2. Informative References

[CANON] Schwartz, E. and B. Kallick, "Generating a canonical prefix encoding", Communications of the ACM Volume 7 Issue 3, pp. 166-169, March 1964, <http://dl.acm.org/citation.cfm?id=363991>.

[CRIME] Rizzo, J. and T. Duong, "The CRIME Attack", September 2012, <https://docs.google.com/a/twist.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1eb6c1b5_3_6>.

[DEFLATE] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[HUFF] Huffman, D., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers Volume 40, Number 9, pp. 1098-1101, September 1952, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4051119>.

[PERF1] Belshe, M., "IETF83: SPDY and What to Consider for HTTP/2.0", March 2012, <http://www.ietf.org/proceedings/83/slides/slides-83-httpbis-3>.

[PERF2] McManus, P., "SPDY: What I Like About You", September 2011, <http://bitsup.blogspot.com/2011/09/spdy-what-i-like-about-you.html>.

[PETAL] Tan, J. and J. Nahata, "PETAL: Preset Encoding Table Information Leakage", April 2013, <http://www.pdl.cmu.edu/PDL-FTP/associated/CMU-PDL-13-106.pdf>.

[SPDY] Belshe, M. and R. Peon, "SPDY Protocol", draft-mbelshe-httpbis-spdy-00 (work in progress), February 2012.

Appendix A. Change Log (to be removed by RFC Editor before publication

A.1. Since draft-ietf-httpbis-header-compression-06

- Updated format to include literal headers that must never be compressed.
- Updated security considerations.
- Moved integer encoding examples to the appendix.
- Updated Huffman table.
- Updated static header table (adding and removing status values).
- Updated examples.

A.2. Since draft-ietf-httpbis-header-compression-05

- Regenerated examples.
- Only one Huffman table for requests and responses.
- Added maximum size for header table, independent of SETTINGS_HEADER_TABLE_SIZE.
- Added pseudo-code for integer decoding.
- Improved examples (removing unnecessary removals).

A.3. Since draft-ietf-httpbis-header-compression-04

- Updated examples: take into account changes in the spec, and show more features.
- Use 'octet' everywhere instead of having both 'byte' and 'octet'.
- Added reference set emptying.
- Editorial changes and clarifications.
- Added "host" header to the static table.
- Ordering for list of values (either NULL- or comma-separated).

A.4. Since draft-ietf-httpbis-header-compression-03

- A large number of editorial changes; changed the description of evicting/adding new entries.
- Removed substitution indexing
- Changed 'initial headers' to 'static headers', as per issue #258
- Merged 'request' and 'response' static headers, as per issue #259
- Changed text to indicate that new headers are added at index 0 and expire from the largest index, as per issue #233

A.5. Since draft-ietf-httpbis-header-compression-02

- Corrected error in integer encoding pseudocode.

A.6. Since draft-ietf-httpbis-header-compression-01

- Refactored of Header Encoding Section: split definitions and processing rule.
- Backward incompatible change: Updated reference set management as per issue #214. This changes how the interaction between the reference set and eviction works. This also changes the working of the reference set in some specific cases.
- Backward incompatible change: modified initial header list, as per issue #188.
- Added example of 32 octets entry structure (issue #191).
- Added Header Set Completion section. Reflowed some text. Clarified some writing which was akward. Added text about duplicate header entry encoding. Clarified some language w.r.t Header Set. Changed x-my-header to mynewheader. Added text in the HeaderEmission section indicating that the application may also be able to free up memory more quickly. Added information in Security Considerations section.

A.7. Since draft-ietf-httpbis-header-compression-00

Fixed bug/omission in integer representation algorithm.

Changed the document title.

Header matching text rewritten.

Changed the definition of header emission.

Changed the name of the setting which dictates how much memory the compression context should use.

Removed "specific use cases" section

Corrected erroneous statement about what index can be contained in one octet

Added descriptions of opcodes

Removed security claims from introduction.

Appendix B. Static Table

The static table consists of an unchangeable ordered list of (name, value) pairs. The first entry in the table is always represented by the index len(header table) + 1, and the last entry in the table is represented by the index len(header table) + len(static table).

The following table lists the pre-defined header fields that make-up the static table.

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
15	accept-charset	
16	accept-encoding	
17	accept-language	
18	accept-ranges	
19	accept	
20	access-control-allow-origin	
21	age	
22	allow	
23	authorization	

24	cache-control	
25	content-disposition	
26	content-encoding	
27	content-language	
28	content-length	
29	content-location	
30	content-range	
31	content-type	
32	cookie	
33	date	
34	etag	
35	expect	
36	expires	
37	from	
38	host	
39	if-match	
40	if-modified-since	
41	if-none-match	
42	if-range	
43	if-unmodified-since	
44	last-modified	
45	link	
46	location	
47	max-forwards	
48	proxy-authenticate	
49	proxy-authorization	
50	range	
51	referer	
52	refresh	
53	retry-after	
54	server	
55	set-cookie	
56	strict-transport-security	
57	transfer-encoding	
58	user-agent	
59	vary	
60	via	
61	www-authenticate	

Table 1: Static Table Entries

The table give the index of each entry in the static table. The full index of each entry, to be used for encoding a reference to this entry, is computed by adding the number of entries in the header table to this index.

Appendix C. Huffman Codes

The following codes are used when encoding string literals with an Huffman coding (see Section 4.1.2).

Each row in the table specifies one Huffman code:

sym: The symbol to be represented. It is the decimal value of an octet, possibly prepended with its ASCII representation. A specific symbol, "EOS", is used to indicate the end of a string literal.

code as bits: The Huffman code for the symbol represented as a base-2 integer.

code as hex: The Huffman code for the symbol, represented as a hexadecimal integer, aligned on the least significant bit.

len: The number of bits for the Huffman code of the symbol.

As an example, the Huffman code for the symbol 48 (corresponding to the ASCII character "0") consists in the 5 bits "0", "0", "1", "0", "1". This corresponds to the value 5 encoded on 5 bits.

sym	code as bits aligned to MSB	code as hex aligned to LSB	len in bits
(0)	|11111111|11111111|11101110|10	3ffffba	[26]
(1)	|11111111|11111111|11101110|11	3ffffbb	[26]
(2)	|11111111|11111111|11101111|00	3ffffbc	[26]
(3)	|11111111|11111111|11101111|01	3ffffbd	[26]
(4)	|11111111|11111111|11101111|10	3ffffbe	[26]
(5)	|11111111|11111111|11101111|11	3ffffbf	[26]
(6)	|11111111|11111111|11110000|00	3ffffc0	[26]
(7)	|11111111|11111111|11110000|01	3ffffc1	[26]
(8)	|11111111|11111111|11110000|10	3ffffc2	[26]
(9)	|11111111|11111111|11110000|11	3ffffc3	[26]
(10)	|11111111|11111111|11110001|00	3ffffc4	[26]
(11)	|11111111|11111111|11110001|01	3ffffc5	[26]
(12)	|11111111|11111111|11110001|10	3ffffc6	[26]
(13)	|11111111|11111111|11110001|11	3ffffc7	[26]
(14)	|11111111|11111111|11110010|00	3ffffc8	[26]
(15)	|11111111|11111111|11110010|01	3ffffc9	[26]
(16)	|11111111|11111111|11110010|10	3ffffca	[26]
(17)	|11111111|11111111|11110010|11	3ffffcb	[26]
(18)	|11111111|11111111|11110011|00	3ffffcc	[26]
(19)	|11111111|11111111|11110011|01	3ffffcd	[26]

```
      ( 20)  |11111111|11111111|11110011|10  3ffffce  [26]
      ( 21)  |11111111|11111111|11110011|11  3ffffcf  [26]
      ( 22)  |11111111|11111111|11110100|00  3ffffd0  [26]
      ( 23)  |11111111|11111111|11110100|01  3ffffd1  [26]
      ( 24)  |11111111|11111111|11110100|10  3ffffd2  [26]
      ( 25)  |11111111|11111111|11110100|11  3ffffd3  [26]
      ( 26)  |11111111|11111111|11110101|00  3ffffd4  [26]
      ( 27)  |11111111|11111111|11110101|01  3ffffd5  [26]
      ( 28)  |11111111|11111111|11110101|10  3ffffd6  [26]
      ( 29)  |11111111|11111111|11110101|11  3ffffd7  [26]
      ( 30)  |11111111|11111111|11110110|00  3ffffd8  [26]
      ( 31)  |11111111|11111111|11110110|01  3ffffd9  [26]
  ' ' ( 32)  |00110                              6  [ 5]
  '!' ( 33)  |11111111|11100                   1ffc  [13]
  '"' ( 34)  |11111000|0                        1f0  [ 9]
  '#' ( 35)  |11111111|111100                  3ffc  [14]
  '$' ( 36)  |11111111|1111100                 7ffc  [15]
  '%' ( 37)  |011110                             1e  [ 6]
  '&' ( 38)  |1100100                            64  [ 7]
  ''' ( 39)  |11111111|11101                   1ffd  [13]
  '(' ( 40)  |11111110|10                       3fa  [10]
  ')' ( 41)  |11111000|1                        1f1  [ 9]
  '*' ( 42)  |11111110|11                       3fb  [10]
  '+' ( 43)  |11111111|00                       3fc  [10]
  ',' ( 44)  |1100101                            65  [ 7]
  '-' ( 45)  |1100110                            66  [ 7]
  '.' ( 46)  |011111                             1f  [ 6]
  '/' ( 47)  |00111                              7  [ 5]
  '0' ( 48)  |0000                               0  [ 4]
  '1' ( 49)  |0001                               1  [ 4]
  '2' ( 50)  |0010                               2  [ 4]
  '3' ( 51)  |01000                              8  [ 5]
  '4' ( 52)  |100000                            20  [ 6]
  '5' ( 53)  |100001                            21  [ 6]
  '6' ( 54)  |100010                            22  [ 6]
  '7' ( 55)  |100011                            23  [ 6]
  '8' ( 56)  |100100                            24  [ 6]
  '9' ( 57)  |100101                            25  [ 6]
  ':' ( 58)  |100110                            26  [ 6]
  ';' ( 59)  |11101100|                          ec  [ 8]
  '<' ( 60)  |11111111|11111110|0              1fffc  [17]
  '=' ( 61)  |100111                            27  [ 6]
  '>' ( 62)  |11111111|1111101                 7ffd  [15]
  '?' ( 63)  |11111111|01                       3fd  [10]
  '@' ( 64)  |11111111|1111110                 7ffe  [15]
  'A' ( 65)  |1100111                            67  [ 7]
  'B' ( 66)  |11101101|                          ed  [ 8]
  'C' ( 67)  |11101110|                          ee  [ 8]
```

```
   'D' ( 68)  |1101000                                68  [ 7]
   'E' ( 69)  |11101111|                              ef  [ 8]
   'F' ( 70)  |1101001                                69  [ 7]
   'G' ( 71)  |1101010                                6a  [ 7]
   'H' ( 72)  |11111001|0                            1f2  [ 9]
   'I' ( 73)  |11110000|                              f0  [ 8]
   'J' ( 74)  |11111001|1                            1f3  [ 9]
   'K' ( 75)  |11111010|0                            1f4  [ 9]
   'L' ( 76)  |11111010|1                            1f5  [ 9]
   'M' ( 77)  |1101011                                6b  [ 7]
   'N' ( 78)  |1101100                                6c  [ 7]
   'O' ( 79)  |11110001|                              f1  [ 8]
   'P' ( 80)  |11110010|                              f2  [ 8]
   'Q' ( 81)  |11111011|0                            1f6  [ 9]
   'R' ( 82)  |11111011|1                            1f7  [ 9]
   'S' ( 83)  |1101101                                6d  [ 7]
   'T' ( 84)  |101000                                 28  [ 6]
   'U' ( 85)  |11110011|                              f3  [ 8]
   'V' ( 86)  |11111100|0                            1f8  [ 9]
   'W' ( 87)  |11111100|1                            1f9  [ 9]
   'X' ( 88)  |11110100|                              f4  [ 8]
   'Y' ( 89)  |11111101|0                            1fa  [ 9]
   'Z' ( 90)  |11111101|1                            1fb  [ 9]
   '[' ( 91)  |11111111|100                          7fc  [11]
   '\' ( 92)  |11111111|11111111|11110110|10      3ffffda  [26]
   ']' ( 93)  |11111111|101                          7fd  [11]
   '^' ( 94)  |11111111|111101                     3ffd  [14]
   '_' ( 95)  |1101110                                6e  [ 7]
   '`' ( 96)  |11111111|11111111|10                3fffe  [18]
   'a' ( 97)  |01001                                   9  [ 5]
   'b' ( 98)  |1101111                                6f  [ 7]
   'c' ( 99)  |01010                                   a  [ 5]
   'd' (100)  |101001                                 29  [ 6]
   'e' (101)  |01011                                   b  [ 5]
   'f' (102)  |1110000                                70  [ 7]
   'g' (103)  |101010                                 2a  [ 6]
   'h' (104)  |101011                                 2b  [ 6]
   'i' (105)  |01100                                   c  [ 5]
   'j' (106)  |11110101|                              f5  [ 8]
   'k' (107)  |11110110|                              f6  [ 8]
   'l' (108)  |101100                                 2c  [ 6]
   'm' (109)  |101101                                 2d  [ 6]
   'n' (110)  |101110                                 2e  [ 6]
   'o' (111)  |01101                                   d  [ 5]
   'p' (112)  |101111                                 2f  [ 6]
   'q' (113)  |11111110|0                            1fc  [ 9]
   'r' (114)  |110000                                 30  [ 6]
   's' (115)  |110001                                 31  [ 6]
```

```
   't' (116)  |01110                                    e  [ 5]
   'u' (117)  |1110001                                 71  [ 7]
   'v' (118)  |1110010                                 72  [ 7]
   'w' (119)  |1110011                                 73  [ 7]
   'x' (120)  |1110100                                 74  [ 7]
   'y' (121)  |1110101                                 75  [ 7]
   'z' (122)  |11110111|                               f7  [ 8]
   '{' (123)  |11111111|11111110|1                  1fffd  [17]
   '|' (124)  |11111111|1100                         ffc  [12]
   '}' (125)  |11111111|11111111|0                  1fffe  [17]
   '~' (126)  |11111111|1101                         ffd  [12]
      (127)  |11111111|11111111|11110110|11  3ffffdb  [26]
      (128)  |11111111|11111111|11110111|00  3ffffdc  [26]
      (129)  |11111111|11111111|11110111|01  3ffffdd  [26]
      (130)  |11111111|11111111|11110111|10  3ffffde  [26]
      (131)  |11111111|11111111|11110111|11  3ffffdf  [26]
      (132)  |11111111|11111111|11111000|00  3ffffe0  [26]
      (133)  |11111111|11111111|11111000|01  3ffffe1  [26]
      (134)  |11111111|11111111|11111000|10  3ffffe2  [26]
      (135)  |11111111|11111111|11111000|11  3ffffe3  [26]
      (136)  |11111111|11111111|11111001|00  3ffffe4  [26]
      (137)  |11111111|11111111|11111001|01  3ffffe5  [26]
      (138)  |11111111|11111111|11111001|10  3ffffe6  [26]
      (139)  |11111111|11111111|11111001|11  3ffffe7  [26]
      (140)  |11111111|11111111|11111010|00  3ffffe8  [26]
      (141)  |11111111|11111111|11111010|01  3ffffe9  [26]
      (142)  |11111111|11111111|11111010|10  3ffffea  [26]
      (143)  |11111111|11111111|11111010|11  3ffffeb  [26]
      (144)  |11111111|11111111|11111011|00  3ffffec  [26]
      (145)  |11111111|11111111|11111011|01  3ffffed  [26]
      (146)  |11111111|11111111|11111011|10  3ffffee  [26]
      (147)  |11111111|11111111|11111011|11  3ffffef  [26]
      (148)  |11111111|11111111|11111100|00  3fffff0  [26]
      (149)  |11111111|11111111|11111100|01  3fffff1  [26]
      (150)  |11111111|11111111|11111100|10  3fffff2  [26]
      (151)  |11111111|11111111|11111100|11  3fffff3  [26]
      (152)  |11111111|11111111|11111101|00  3fffff4  [26]
      (153)  |11111111|11111111|11111101|01  3fffff5  [26]
      (154)  |11111111|11111111|11111101|10  3fffff6  [26]
      (155)  |11111111|11111111|11111101|11  3fffff7  [26]
      (156)  |11111111|11111111|11111110|00  3fffff8  [26]
      (157)  |11111111|11111111|11111110|01  3fffff9  [26]
      (158)  |11111111|11111111|11111110|10  3ffffffa  [26]
      (159)  |11111111|11111111|11111110|11  3ffffffb  [26]
      (160)  |11111111|11111111|11111111|00  3ffffffc  [26]
      (161)  |11111111|11111111|11111111|01  3ffffffd  [26]
      (162)  |11111111|11111111|11111111|10  3ffffffe  [26]
      (163)  |11111111|11111111|11111111|11  3fffffff  [26]
```

```
(164)  |11111111|11111111|11000000|0   1ffff80  [25]
(165)  |11111111|11111111|11000000|1   1ffff81  [25]
(166)  |11111111|11111111|11000001|0   1ffff82  [25]
(167)  |11111111|11111111|11000001|1   1ffff83  [25]
(168)  |11111111|11111111|11000010|0   1ffff84  [25]
(169)  |11111111|11111111|11000010|1   1ffff85  [25]
(170)  |11111111|11111111|11000011|0   1ffff86  [25]
(171)  |11111111|11111111|11000011|1   1ffff87  [25]
(172)  |11111111|11111111|11000100|0   1ffff88  [25]
(173)  |11111111|11111111|11000100|1   1ffff89  [25]
(174)  |11111111|11111111|11000101|0   1ffff8a  [25]
(175)  |11111111|11111111|11000101|1   1ffff8b  [25]
(176)  |11111111|11111111|11000110|0   1ffff8c  [25]
(177)  |11111111|11111111|11000110|1   1ffff8d  [25]
(178)  |11111111|11111111|11000111|0   1ffff8e  [25]
(179)  |11111111|11111111|11000111|1   1ffff8f  [25]
(180)  |11111111|11111111|11001000|0   1ffff90  [25]
(181)  |11111111|11111111|11001000|1   1ffff91  [25]
(182)  |11111111|11111111|11001001|0   1ffff92  [25]
(183)  |11111111|11111111|11001001|1   1ffff93  [25]
(184)  |11111111|11111111|11001010|0   1ffff94  [25]
(185)  |11111111|11111111|11001010|1   1ffff95  [25]
(186)  |11111111|11111111|11001011|0   1ffff96  [25]
(187)  |11111111|11111111|11001011|1   1ffff97  [25]
(188)  |11111111|11111111|11001100|0   1ffff98  [25]
(189)  |11111111|11111111|11001100|1   1ffff99  [25]
(190)  |11111111|11111111|11001101|0   1ffff9a  [25]
(191)  |11111111|11111111|11001101|1   1ffff9b  [25]
(192)  |11111111|11111111|11001110|0   1ffff9c  [25]
(193)  |11111111|11111111|11001110|1   1ffff9d  [25]
(194)  |11111111|11111111|11001111|0   1ffff9e  [25]
(195)  |11111111|11111111|11001111|1   1ffff9f  [25]
(196)  |11111111|11111111|11010000|0   1ffffa0  [25]
(197)  |11111111|11111111|11010000|1   1ffffa1  [25]
(198)  |11111111|11111111|11010001|0   1ffffa2  [25]
(199)  |11111111|11111111|11010001|1   1ffffa3  [25]
(200)  |11111111|11111111|11010010|0   1ffffa4  [25]
(201)  |11111111|11111111|11010010|1   1ffffa5  [25]
(202)  |11111111|11111111|11010011|0   1ffffa6  [25]
(203)  |11111111|11111111|11010011|1   1ffffa7  [25]
(204)  |11111111|11111111|11010100|0   1ffffa8  [25]
(205)  |11111111|11111111|11010100|1   1ffffa9  [25]
(206)  |11111111|11111111|11010101|0   1ffffaa  [25]
(207)  |11111111|11111111|11010101|1   1ffffab  [25]
(208)  |11111111|11111111|11010110|0   1ffffac  [25]
(209)  |11111111|11111111|11010110|1   1ffffad  [25]
(210)  |11111111|11111111|11010111|0   1ffffae  [25]
(211)  |11111111|11111111|11010111|1   1ffffaf  [25]
```

```
      (212)  |11111111|11111111|11011000|0   1fffffb0  [25]
      (213)  |11111111|11111111|11011000|1   1fffffb1  [25]
      (214)  |11111111|11111111|11011001|0   1fffffb2  [25]
      (215)  |11111111|11111111|11011001|1   1fffffb3  [25]
      (216)  |11111111|11111111|11011010|0   1fffffb4  [25]
      (217)  |11111111|11111111|11011010|1   1fffffb5  [25]
      (218)  |11111111|11111111|11011011|0   1fffffb6  [25]
      (219)  |11111111|11111111|11011011|1   1fffffb7  [25]
      (220)  |11111111|11111111|11011100|0   1fffffb8  [25]
      (221)  |11111111|11111111|11011100|1   1fffffb9  [25]
      (222)  |11111111|11111111|11011101|0   1fffffba  [25]
      (223)  |11111111|11111111|11011101|1   1fffffbb  [25]
      (224)  |11111111|11111111|11011110|0   1fffffbc  [25]
      (225)  |11111111|11111111|11011110|1   1fffffbd  [25]
      (226)  |11111111|11111111|11011111|0   1fffffbe  [25]
      (227)  |11111111|11111111|11011111|1   1fffffbf  [25]
      (228)  |11111111|11111111|11100000|0   1fffffc0  [25]
      (229)  |11111111|11111111|11100000|1   1fffffc1  [25]
      (230)  |11111111|11111111|11100001|0   1fffffc2  [25]
      (231)  |11111111|11111111|11100001|1   1fffffc3  [25]
      (232)  |11111111|11111111|11100010|0   1fffffc4  [25]
      (233)  |11111111|11111111|11100010|1   1fffffc5  [25]
      (234)  |11111111|11111111|11100011|0   1fffffc6  [25]
      (235)  |11111111|11111111|11100011|1   1fffffc7  [25]
      (236)  |11111111|11111111|11100100|0   1fffffc8  [25]
      (237)  |11111111|11111111|11100100|1   1fffffc9  [25]
      (238)  |11111111|11111111|11100101|0   1fffffca  [25]
      (239)  |11111111|11111111|11100101|1   1fffffcb  [25]
      (240)  |11111111|11111111|11100110|0   1fffffcc  [25]
      (241)  |11111111|11111111|11100110|1   1fffffcd  [25]
      (242)  |11111111|11111111|11100111|0   1fffffce  [25]
      (243)  |11111111|11111111|11100111|1   1fffffcf  [25]
      (244)  |11111111|11111111|11101000|0   1fffffd0  [25]
      (245)  |11111111|11111111|11101000|1   1fffffd1  [25]
      (246)  |11111111|11111111|11101001|0   1fffffd2  [25]
      (247)  |11111111|11111111|11101001|1   1fffffd3  [25]
      (248)  |11111111|11111111|11101010|0   1fffffd4  [25]
      (249)  |11111111|11111111|11101010|1   1fffffd5  [25]
      (250)  |11111111|11111111|11101011|0   1fffffd6  [25]
      (251)  |11111111|11111111|11101011|1   1fffffd7  [25]
      (252)  |11111111|11111111|11101100|0   1fffffd8  [25]
      (253)  |11111111|11111111|11101100|1   1fffffd9  [25]
      (254)  |11111111|11111111|11101101|0   1fffffda  [25]
      (255)  |11111111|11111111|11101101|1   1fffffdb  [25]
  EOS (256)  |11111111|11111111|11101110|0   1fffffdc  [25]
```

Appendix D. Examples

A number of examples are worked through here, covering integer encoding, header field representation, and the encoding of whole sets of header fields, for both requests and responses, and with and without Huffman coding.

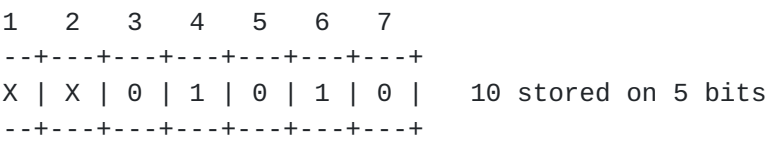
D.1. Integer Representation Examples

This section shows the representation of integer values in details (see Section 4.1.1).

D.1.1. Example 1: Encoding 10 using a 5-bit prefix

The value 10 is to be encoded with a 5-bit prefix.

- 10 is less than 31 (2^5 - 1) and is represented using the 5-bit prefix.



D.1.2. Example 2: Encoding 1337 using a 5-bit prefix

The value I=1337 is to be encoded with a 5-bit prefix.

1337 is greater than 31 (2^5 - 1).

The 5-bit prefix is filled with its max value (31).

I = 1337 - (2^5 - 1) = 1306.

I (1306) is greater than or equal to 128, the while loop body executes:

I % 128 == 26

26 + 128 == 154

154 is encoded in 8 bits as: 10011010

I is set to 10 (1306 / 128 == 10)

I is no longer greater than or equal to 128, the while loop terminates.

I, now 10, is encoded on 8 bits as: 00001010.

The process ends.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| X | X | X | 1 | 1 | 1 | 1 | 1 |  Prefix = 31, I = 1306
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |  1306>=128, encode(154), I=1306/128
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |  10<128, encode(10), done
+---+---+---+---+---+---+---+---+
```

D.1.3. Example 3: Encoding 42 starting at an octet-boundary

The value 42 is to be encoded starting at an octet-boundary. This implies that a 8-bit prefix is used.

o 42 is less than 255 (2^8 - 1) and is represented using the 8-bit prefix.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |   42 stored on 8 bits
+---+---+---+---+---+---+---+---+
```

D.2. Header Field Representation Examples

This section shows several independent representation examples.

D.2.1. Literal Header Field with Indexing

The header field representation uses a literal name and a literal value.

Header set to encode:

custom-key: custom-header

Reference set: empty.

Hex dump of encoded data:

```
400a 6375 7374 6f6d 2d6b 6579 0d63 7573 | @.custom-key.cus
746f 6d2d 6865 6164 6572                | tom-header
```

Decoding process:

```
40                                      | == Literal indexed ==
0a                                      |   Literal name (len = 10)
6375 7374 6f6d 2d6b 6579                | custom-key
0d                                      |   Literal value (len = 13)
6375 7374 6f6d 2d68 6561 6465 72        | custom-header
                                        | -> custom-key: custom-head\
                                        |   er
```

Header Table (after decoding):

```
[  1] (s =  55) custom-key: custom-header
      Table size:  55
```

Decoded header set:

```
custom-key: custom-header
```

D.2.2. Literal Header Field without Indexing

The header field representation uses an indexed name and a literal value.

Header set to encode:

```
:path: /sample/path
```

Reference set: empty.

Hex dump of encoded data:

```
040c 2f73 616d 706c 652f 7061 7468       | ../sample/path
```

Decoding process:

```
04                                      | == Literal not indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :path
0c                                      |   Literal value (len = 12)
2f73 616d 706c 652f 7061 7468           | /sample/path
                                        | -> :path: /sample/path
```

Header table (after decoding): empty.

Decoded header set:

:path: /sample/path

D.2.3. Indexed Header Field

The header field representation uses an indexed header field, from the static table. Upon using it, the static table entry is copied into the header table.

Header set to encode:

:method: GET

Reference set: empty.

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
```

Header Table (after decoding):

```
[  1] (s =  42) :method: GET
      Table size:  42
```

Decoded header set:

```
:method: GET
```

D.2.4. Indexed Header Field from Static Table

The header field representation uses an indexed header field, from the static table. In this example, the SETTINGS_HEADER_TABLE_SIZE is set to 0, therefore, the entry is not copied into the header table.

Header set to encode:

```
:method: GET
```

Reference set: empty.

Hex dump of encoded data:

```
82                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
```

Header table (after decoding): empty.

Decoded header set:

```
:method: GET
```

D.3. Request Examples without Huffman

This section shows several consecutive header sets, corresponding to HTTP requests, on the same connection.

D.3.1. First request

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
8287 8644 0f77 7777 2e65 7861 6d70 6c65 | ...D.www.example
2e63 6f6d                               | .com
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: http
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :path: /
44                                      | == Literal indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :authority
0f                                      |   Literal value (len = 15)
7777 772e 6578 616d 706c 652e 636f 6d   | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Header Table (after decoding):

```
[  1] (s =  57) :authority: www.example.com
[  2] (s =  38) :path: /
[  3] (s =  43) :scheme: http
[  4] (s =  42) :method: GET
      Table size: 180
```

Decoded header set:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

D.3.2. Second request

This request takes advantage of the differential encoding of header sets.

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache
```

Reference set:

```
[  1] :authority: www.example.com
[  2] :path: /
[  3] :scheme: http
[  4] :method: GET
```

Hex dump of encoded data:

```
5c08 6e6f 2d63 6163 6865                | \.no-cache
```

Decoding process:

```
5c                                      | == Literal indexed ==
                                        |   Indexed name (idx = 28)
                                        |     cache-control
08                                      |   Literal value (len = 8)
6e6f 2d63 6163 6865                     | no-cache
                                        | -> cache-control: no-cache
```

Header Table (after decoding):

```
[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
[  3] (s =  38) :path: /
[  4] (s =  43) :scheme: http
[  5] (s =  42) :method: GET
      Table size: 233
```

Decoded header set:

```
cache-control: no-cache
:authority: www.example.com
:path: /
:scheme: http
:method: GET
```

D.3.3. Third request

This request has not enough headers in common with the previous request to take advantage of the differential encoding. Therefore, the reference set is emptied before encoding the header fields.

Header set to encode:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

Reference set:

```
[  1] cache-control: no-cache
[  2] :authority: www.example.com
[  3] :path: /
[  4] :scheme: http
[  5] :method: GET
```

Hex dump of encoded data:

```
3085 8c8b 8440 0a63 7573 746f 6d2d 6b65 | 0....@.custom-ke
790c 6375 7374 6f6d 2d76 616c 7565      | y.custom-value
```

Decoding process:

```
30                                      | == Empty reference set ==
                                        |   idx = 0
                                        |   flag = 1
85                                      | == Indexed - Add ==
                                        |   idx = 5
                                        | -> :method: GET
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | -> :scheme: https
8b                                      | == Indexed - Add ==
                                        |   idx = 11
                                        | -> :path: /index.html
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> :authority: www.example\
                                        |   .com
40                                      | == Literal indexed ==
0a                                      |   Literal name (len = 10)
6375 7374 6f6d 2d6b 6579                | custom-key
0c                                      |   Literal value (len = 12)
6375 7374 6f6d 2d76 616c 7565           | custom-value
                                        | -> custom-key: custom-valu\
                                        |   e
```

Header Table (after decoding):

```
[  1] (s =  54) custom-key: custom-value
[  2] (s =  48) :path: /index.html
[  3] (s =  44) :scheme: https
[  4] (s =  53) cache-control: no-cache
[  5] (s =  57) :authority: www.example.com
[  6] (s =  38) :path: /
[  7] (s =  43) :scheme: http
[  8] (s =  42) :method: GET
      Table size: 379
```

Decoded header set:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

D.4. Request Examples with Huffman

This section shows the same examples as the previous section, but using Huffman encoding for the literal values.

D.4.1. First request

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
8287 8644 8ce7 cf9b ebe8 9b6f b16f a9b6 | ...D.......o.o..
ff                                      | .
```

Decoding process:

```
82                                      | == Indexed - Add ==
                                        |   idx = 2
                                        | -> :method: GET
87                                      | == Indexed - Add ==
                                        |   idx = 7
                                        | -> :scheme: http
86                                      | == Indexed - Add ==
                                        |   idx = 6
                                        | -> :path: /
44                                      | == Literal indexed ==
                                        |   Indexed name (idx = 4)
                                        |     :authority
8c                                      |   Literal value (len = 15)
                                        |     Huffman encoded:
e7cf 9beb e89b 6fb1 6fa9 b6ff           | ......o.o...
                                        |     Decoded:
                                        | www.example.com
                                        | -> :authority: www.example\
                                        |   .com
```

Header Table (after decoding):

```
[  1] (s =  57) :authority: www.example.com
[  2] (s =  38) :path: /
[  3] (s =  43) :scheme: http
[  4] (s =  42) :method: GET
      Table size: 180
```

Decoded header set:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
```

D.4.2. Second request

This request takes advantage of the differential encoding of header sets.

Header set to encode:

```
:method: GET
:scheme: http
:path: /
:authority: www.example.com
cache-control: no-cache
```

Reference set:

```
[  1] :authority: www.example.com
[  2] :path: /
[  3] :scheme: http
[  4] :method: GET
```

Hex dump of encoded data:

```
5c86 b9b9 9495 56bf                     | \.....V.
```

Decoding process:

```
   5c                                      | == Literal indexed ==
                                           |   Indexed name (idx = 28)
                                           |     cache-control
   86                                      |   Literal value (len = 8)
                                           |     Huffman encoded:
   b9b9 9495 56bf                          | ....V.
                                           |     Decoded:
                                           | no-cache
                                           | -> cache-control: no-cache
```

Header Table (after decoding):

```
[  1] (s =  53) cache-control: no-cache
[  2] (s =  57) :authority: www.example.com
[  3] (s =  38) :path: /
[  4] (s =  43) :scheme: http
[  5] (s =  42) :method: GET
      Table size: 233
```

Decoded header set:

```
cache-control: no-cache
:authority: www.example.com
:path: /
:scheme: http
:method: GET
```

D.4.3. Third request

This request has not enough headers in common with the previous request to take advantage of the differential encoding. Therefore, the reference set is emptied before encoding the header fields.

Header set to encode:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

Reference set:

```
   [  1] cache-control: no-cache
   [  2] :authority: www.example.com
   [  3] :path: /
   [  4] :scheme: http
   [  5] :method: GET
```

Hex dump of encoded data:

```
   3085 8c8b 8440 8857 1c5c db73 7b2f af89 | 0....@.W.\.s{/..
   571c 5cdb 7372 4d9c 57                  | W.\.srM.W
```

Decoding process:

```
   30                                      | == Empty reference set ==
                                           |   idx = 0
                                           |   flag = 1
   85                                      | == Indexed - Add ==
                                           |   idx = 5
                                           | -> :method: GET
   8c                                      | == Indexed - Add ==
                                           |   idx = 12
                                           | -> :scheme: https
   8b                                      | == Indexed - Add ==
                                           |   idx = 11
                                           | -> :path: /index.html
   84                                      | == Indexed - Add ==
                                           |   idx = 4
                                           | -> :authority: www.example\
                                           |   .com
   40                                      | == Literal indexed ==
   88                                      |   Literal name (len = 10)
                                           |     Huffman encoded:
   571c 5cdb 737b 2faf                     | W.\.s{/.
                                           |     Decoded:
                                           | custom-key
   89                                      |   Literal value (len = 12)
                                           |     Huffman encoded:
   571c 5cdb 7372 4d9c 57                  | W.\.srM.W
                                           |     Decoded:
                                           | custom-value
                                           | -> custom-key: custom-valu\
                                           |   e
```

Header Table (after decoding):

```
[  1] (s =  54) custom-key: custom-value
[  2] (s =  48) :path: /index.html
[  3] (s =  44) :scheme: https
[  4] (s =  53) cache-control: no-cache
[  5] (s =  57) :authority: www.example.com
[  6] (s =  38) :path: /
[  7] (s =  43) :scheme: http
[  8] (s =  42) :method: GET
      Table size: 379
```

Decoded header set:

```
:method: GET
:scheme: https
:path: /index.html
:authority: www.example.com
custom-key: custom-value
```

D.5. Response Examples without Huffman

This section shows several consecutive header sets, corresponding to HTTP responses, on the same connection. SETTINGS_HEADER_TABLE_SIZE is set to the value of 256 octets, causing some evictions to occur.

D.5.1. First response

Header set to encode:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
4803 3330 3259 0770 7269 7661 7465 631d | H.302Y.privatec.
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 5471 1768 |  20:13:21 GMTq.h
7474 7073 3a2f 2f77 7777 2e65 7861 6d70 | ttps://www.examp
6c65 2e63 6f6d                          | le.com
```

Decoding process:

```
48                                      | == Literal indexed ==
                                        |   Indexed name (idx = 8)
                                        |     :status
03                                      |   Literal value (len = 3)
3330 32                                 | 302
                                        | -> :status: 302
59                                      | == Literal indexed ==
                                        |   Indexed name (idx = 25)
                                        |     cache-control
07                                      |   Literal value (len = 7)
7072 6976 6174 65                       | private
                                        | -> cache-control: private
63                                      | == Literal indexed ==
                                        |   Indexed name (idx = 35)
                                        |     date
1d                                      |   Literal value (len = 29)
4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
2032 303a 3133 3a32 3120 474d 54        |  20:13:21 GMT
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:21 GMT
71                                      | == Literal indexed ==
                                        |   Indexed name (idx = 49)
                                        |     location
17                                      |   Literal value (len = 23)
6874 7470 733a 2f2f 7777 772e 6578 616d | https://www.exam
706c 652e 636f 6d                       | ple.com
                                        | -> location: https://www.e\
                                        |   xample.com
```

Header Table (after decoding):

```
[  1] (s =  63) location: https://www.example.com
[  2] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] (s =  52) cache-control: private
[  4] (s =  42) :status: 302
      Table size: 222
```

Decoded header set:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

D.5.2. Second response

The (":status", "302") header field is evicted from the header table to free space to allow adding the (":status", "200") header field, copied from the static table into the header table. The (":status", "302") header field doesn't need to be removed from the reference set as it is evicted from the header table.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set:

```
[  1] location: https://www.example.com
[  2] date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] cache-control: private
[  4] :status: 302
```

Hex dump of encoded data:

```
8c                                      | .
```

Decoding process:

```
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | - evict: :status: 302
                                        | -> :status: 200
```

Header Table (after decoding):

```
[  1] (s =  42) :status: 200
[  2] (s =  63) location: https://www.example.com
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] (s =  52) cache-control: private
      Table size: 222
```

Decoded header set:

```
:status: 200
location: https://www.example.com
date: Mon, 21 Oct 2013 20:13:21 GMT
cache-control: private
```

D.5.3. Third response

Several header fields are evicted from the header table during the processing of this header set. Before evicting a header belonging to the reference set, it is emitted, by coding it twice as an Indexed Representation. The first representation removes the header field from the reference set, the second one adds it again to the reference set, also emitting it.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
location: https://www.example.com
content-encoding: gzip
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

Reference set:

```
[  1] :status: 200
[  2] location: https://www.example.com
[  3] date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] cache-control: private
```

Hex dump of encoded data:

```
8484 431d 4d6f 6e2c 2032 3120 4f63 7420 | ..C.Mon, 21 Oct
3230 3133 2032 303a 3133 3a32 3220 474d | 2013 20:13:22 GM
545e 0467 7a69 7084 8483 837b 3866 6f6f | T^.gzip....{8foo
3d41 5344 4a4b 4851 4b42 5a58 4f51 5745 | =ASDJKHQKBZXOQWE
4f50 4955 4158 5157 454f 4955 3b20 6d61 | OPIUAXQWEOIU; ma
782d 6167 653d 3336 3030 3b20 7665 7273 | x-age=3600; vers
696f 6e3d 31                            | ion=1
```

Decoding process:

```
84                                      | == Indexed - Remove ==
                                        |   idx = 4
```

```
                                          | -> cache-control: private
   84                                     | == Indexed - Add ==
                                          |   idx = 4
                                          | -> cache-control: private
   43                                     | == Literal indexed ==
                                          |   Indexed name (idx = 3)
                                          |     date
   1d                                     |   Literal value (len = 29)
   4d6f 6e2c 2032 3120 4f63 7420 3230 3133 | Mon, 21 Oct 2013
   2032 303a 3133 3a32 3220 474d 54       |  20:13:22 GMT
                                          | - evict: cache-control: pr\
                                          |   ivate
                                          | -> date: Mon, 21 Oct 2013 \
                                          |   20:13:22 GMT
   5e                                     | == Literal indexed ==
                                          |   Indexed name (idx = 30)
                                          |     content-encoding
   04                                     |   Literal value (len = 4)
   677a 6970                              | gzip
                                          | - evict: date: Mon, 21 Oct\
                                          |    2013 20:13:21 GMT
                                          | -> content-encoding: gzip
   84                                     | == Indexed - Remove ==
                                          |   idx = 4
                                          | -> location: https://www.e\
                                          |   xample.com
   84                                     | == Indexed - Add ==
                                          |   idx = 4
                                          | -> location: https://www.e\
                                          |   xample.com
   83                                     | == Indexed - Remove ==
                                          |   idx = 3
                                          | -> :status: 200
   83                                     | == Indexed - Add ==
                                          |   idx = 3
                                          | -> :status: 200
   7b                                     | == Literal indexed ==
                                          |   Indexed name (idx = 59)
                                          |     set-cookie
   38                                     |   Literal value (len = 56)
   666f 6f3d 4153 444a 4b48 514b 425a 584f | foo=ASDJKHQKBZXO
   5157 454f 5049 5541 5851 5745 4f49 553b | QWEOPIUAXQWEOIU;
   206d 6178 2d61 6765 3d33 3630 303b 2076 |  max-age=3600; v
   6572 7369 6f6e 3d31                    | ersion=1
                                          | - evict: location: https:/\
                                          |   /www.example.com
                                          | - evict: :status: 200
                                          | -> set-cookie: foo=ASDJKHQ\
```

```
                                        |   KBZXOQWEOPIUAXQWEOIU; ma\
                                        |   x-age=3600; version=1
```

Header Table (after decoding):

```
[  1] (s =  98) set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age\
                =3600; version=1
[  2] (s =  52) content-encoding: gzip
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:22 GMT
      Table size: 215
```

Decoded header set:

```
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
content-encoding: gzip
location: https://www.example.com
:status: 200
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

D.6. Response Examples with Huffman

This section shows the same examples as the previous section, but using Huffman encoding for the literal values. The eviction mechanism uses the length of the decoded literal values, so the same evictions occurs as in the previous section.

D.6.1. First response

Header set to encode:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set: empty.

Hex dump of encoded data:

```
4882 4017 5985 bf06 724b 9763 93d6 dbb2 | H.@.Y...rK.c....
9884 de2a 7188 0506 2098 5131 09b5 6ba3 | ...*q... .Q1..k.
7191 adce bf19 8e7e 7cf9 bebe 89b6 fb16 | q.......|.......
fa9b 6f                                 | ..o
```

Decoding process:

```
   48                                      | == Literal indexed ==
                                           |   Indexed name (idx = 8)
                                           |     :status
   82                                      |   Literal value (len = 3)
                                           |     Huffman encoded:
   4017                                    | @.
                                           |     Decoded:
                                           | 302
                                           | -> :status: 302
   59                                      | == Literal indexed ==
                                           |   Indexed name (idx = 25)
                                           |     cache-control
   85                                      |   Literal value (len = 7)
                                           |     Huffman encoded:
   bf06 724b 97                            | ..rK.
                                           |     Decoded:
                                           | private
                                           | -> cache-control: private
   63                                      | == Literal indexed ==
                                           |   Indexed name (idx = 35)
                                           |     date
   93                                      |   Literal value (len = 29)
                                           |     Huffman encoded:
   d6db b298 84de 2a71 8805 0620 9851 3109 | ......*q... .Q1.
   b56b a3                                 | .k.
                                           |     Decoded:
                                           | Mon, 21 Oct 2013 20:13:21 \
                                           | GMT
                                           | -> date: Mon, 21 Oct 2013 \
                                           |   20:13:21 GMT
   71                                      | == Literal indexed ==
                                           |   Indexed name (idx = 49)
                                           |     location
   91                                      |   Literal value (len = 23)
                                           |     Huffman encoded:
   adce bf19 8e7e 7cf9 bebe 89b6 fb16 fa9b | ......|.........
   6f                                      | o
                                           |     Decoded:
                                           | https://www.example.com
                                           | -> location: https://www.e\
                                           |   xample.com
```

   Header Table (after decoding):

```
[  1] (s =  63) location: https://www.example.com
[  2] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] (s =  52) cache-control: private
[  4] (s =  42) :status: 302
      Table size: 222
```

Decoded header set:

```
:status: 302
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

D.6.2. Second response

The (":status", "302") header field is evicted from the header table to free space to allow adding the (":status", "200") header field, copied from the static table into the header table. The (":status", "302") header field doesn't need to be removed from the reference set as it is evicted from the header table.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:21 GMT
location: https://www.example.com
```

Reference set:

```
[  1] location: https://www.example.com
[  2] date: Mon, 21 Oct 2013 20:13:21 GMT
[  3] cache-control: private
[  4] :status: 302
```

Hex dump of encoded data:

```
8c                                      | .
```

Decoding process:

```
8c                                      | == Indexed - Add ==
                                        |   idx = 12
                                        | - evict: :status: 302
                                        | -> :status: 200
```

Header Table (after decoding):

```
[  1] (s =  42) :status: 200
[  2] (s =  63) location: https://www.example.com
[  3] (s =  65) date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] (s =  52) cache-control: private
      Table size: 222
```

Decoded header set:

```
:status: 200
location: https://www.example.com
date: Mon, 21 Oct 2013 20:13:21 GMT
cache-control: private
```

D.6.3. Third response

Several header fields are evicted from the header table during the processing of this header set. Before evicting a header belonging to the reference set, it is emitted, by coding it twice as an Indexed Representation. The first representation removes the header field from the reference set, the second one adds it again to the reference set, also emitting it.

Header set to encode:

```
:status: 200
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
location: https://www.example.com
content-encoding: gzip
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

Reference set:

```
[  1] :status: 200
[  2] location: https://www.example.com
[  3] date: Mon, 21 Oct 2013 20:13:21 GMT
[  4] cache-control: private
```

Hex dump of encoded data:

```
8484 4393 d6db b298 84de 2a71 8805 0620 | ..C.......*q...
9851 3111 b56b a35e 84ab dd97 ff84 8483 | .Q1..k.^........
837b b1e0 d6cf 9f6e 8f9f d3e5 f6fa 76fe | .{.....n......v.
fd3c 7edf 9eff 1f2f 0f3c fe9f 6fcf 7f8f | ......./....o...
879f 61ad 4f4c c9a9 73a2 200e c372 5e18 | ..a.OL..s. ..r^.
b1b7 4e3f                               | ..N?
```

Decoding process:

```
84                                      | == Indexed - Remove ==
                                        |   idx = 4
                                        | -> cache-control: private
84                                      | == Indexed - Add ==
                                        |   idx = 4
                                        | -> cache-control: private
43                                      | == Literal indexed ==
                                        |   Indexed name (idx = 3)
                                        |     date
93                                      |   Literal value (len = 29)
                                        |     Huffman encoded:
d6db b298 84de 2a71 8805 0620 9851 3111 | ......*q... .Q1.
b56b a3                                 | .k.
                                        |     Decoded:
                                        | Mon, 21 Oct 2013 20:13:22 \
                                        | GMT
                                        | - evict: cache-control: pr\
                                        |   ivate
                                        | -> date: Mon, 21 Oct 2013 \
                                        |   20:13:22 GMT
5e                                      | == Literal indexed ==
                                        |   Indexed name (idx = 30)
                                        |     content-encoding
84                                      |   Literal value (len = 4)
                                        |     Huffman encoded:
abdd 97ff                               | ....
                                        |     Decoded:
                                        | gzip
```

```
                                          | - evict: date: Mon, 21 Oct\
                                          |    2013 20:13:21 GMT
                                          | -> content-encoding: gzip
   84                                     | == Indexed - Remove ==
                                          |   idx = 4
                                          | -> location: https://www.e\
                                          |   xample.com
   84                                     | == Indexed - Add ==
                                          |   idx = 4
                                          | -> location: https://www.e\
                                          |   xample.com
   83                                     | == Indexed - Remove ==
                                          |   idx = 3
                                          | -> :status: 200
   83                                     | == Indexed - Add ==
                                          |   idx = 3
                                          | -> :status: 200
   7b                                     | == Literal indexed ==
                                          |   Indexed name (idx = 59)
                                          |     set-cookie
   b1                                     |   Literal value (len = 56)
                                          |     Huffman encoded:
   e0d6 cf9f 6e8f 9fd3 e5f6 fa76 fefd 3c7e | ....n......v....
   df9e ff1f 2f0f 3cfe 9f6f cf7f 8f87 9f61 | ..../....o.....a
   ad4f 4cc9 a973 a220 0ec3 725e 18b1 b74e | .OL..s. ..r^...N
   3f                                     | ?
                                          |     Decoded:
                                          | foo=ASDJKHQKBZXOQWEOPIUAXQ\
                                          | WEOIU; max-age=3600; versi\
                                          | on=1
                                          | - evict: location: https:/\
                                          |   /www.example.com
                                          | - evict: :status: 200
                                          | -> set-cookie: foo=ASDJKHQ\
                                          |   KBZXOQWEOPIUAXQWEOIU; ma\
                                          |   x-age=3600; version=1


   Header Table (after decoding):

   [  1] (s =  98) set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age\
                    =3600; version=1
   [  2] (s =  52) content-encoding: gzip
   [  3] (s =  65) date: Mon, 21 Oct 2013 20:13:22 GMT
         Table size: 215
```

Decoded header set:

```
cache-control: private
date: Mon, 21 Oct 2013 20:13:22 GMT
content-encoding: gzip
location: https://www.example.com
:status: 200
set-cookie: foo=ASDJKHQKBZXOQWEOPIUAXQWEOIU; max-age=3600; version=1
```

Authors' Addresses

Roberto Peon
Google, Inc

EMail: fenix@google.com

Herve Ruellan
Canon CRF

EMail: herve.ruellan@crf.canon.fr